CPEN 311: Digital Systems Design

Power Example ($p=x^a$)

1

Development Process

1. Problem statement
2. Interface specification
   – Ports, timing diagram, invariants
3. Test suite, part 1
   – Black-box tests (ie, independent of exact implementation)
4. Algorithm definition and circuit diagram
   – Both datapath and control signals
   – For complex problems, divide into modules (and use hierarchy!)
5. Test suite, part 2
   – Clear-box tests (ie, implementation-dependent)
6. RTL implementation
   – Each module needs a unit testbench (testing only that module)
7. Verification (simulate with ModelSim)
   – Fix bugs until all modules pass unit tests
   – Fix bugs until full design passes all testsuites
   – Add a test for each bug not caught by unit tests, testsuite
8. Pre-fabrication and validation (synthesize and test with Quartus)
   – If constraints (timing, resources) violated, go back to step 4 and re-do
9. Production!

2

Development Challenges

- FPGA design is cheap
- Fabricating a custom chip (ASIC) is very costly
  - Upwards of $1million
- Using a buggy FPGA or ASIC design in a system is extremely costly
  - System redesign
  - Shipped to customers? Failures, recalls, liability
- Verification is often the longest step in design process
  - Labour → very costly
  - Needed to avoid above costs (fabrication, system redesign, customer recalls)
- Why write tests first?
  - Think about requirements harder ➔ get it right early

3

Example – Compute x^a

Problem statement
  Inputs: x, a
  Outputs: $p = x^a$
  For unsigned integers a, x, p

Decisions
  Width of x, a, p in bits?
    32 bits for x, p   5 bits for a

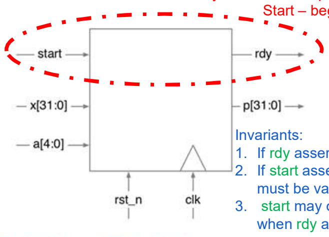
Combinational circuit? (large circuit, possible for small bit widths)
  or
Sequential circuit? (takes multiple clock cycles). ✓

4

Interface (I/Os, invariants)

Interface (specify the I/Os)

Rdy – another computation can start
Start – begin computation

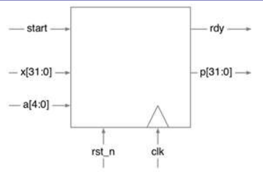


Invariants:
1. If rdy asserted, p is correct
2. If start asserted, x and a must be valid
3. start may only be asserted when rdy asserted

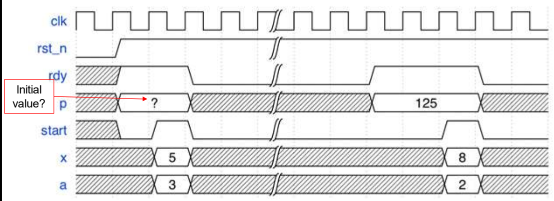
```
module pow(input logic clk, input logic rst_n
           input logic start, input logic [31:0] x, input logic [4:0] a,
           output logic rdy, output logic [31:0] p);
    // TBD
endmodule: pow
```

5

Interface (timing)



Check invariants:
1. If rdy asserted, p is correct
2. If start asserted, x and a must be valid
3. start may only be asserted when rdy asserted
   How do we initialize rdy=1?



6

Test suite, part 1

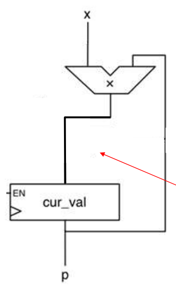
```
module tb_pow();
    logic clk;
    logic rst_n;

    logic start;
    logic [31:0] x;
    logic [4:0] a;

    logic rdy;
    logic [31:0] p;

    pow dut(.*);

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
```

```
module pow(input logic clk, input logic rst_n
           input logic start, input logic [31:0] x, input logic [4:0] a,
           output logic rdy, output logic [31:0] p);
    // TBD
endmodule: pow
```

```
    initial begin
        rst_n <= 1'b0;
        #50;
        rst_n <= 1'b1;

        start <= 1'b0;
        #50;

        assert(rdy === 1'b1);

        x <= 32'd5;
        a <= 5'd2;
        start <= 1'b1;

        #10;
        start <= 1'b0;
        assert(rdy === 1'b0);

        @(posedge rdy);

        assert (p === 32'd25);

        #1000;

        assert(p === 32'd25);

        assert(rdy === 1'b1);

        // rest of our tests here

        $stop;
    end
endmodule: tb_pow
```

7

RTL Implementation (circuit + unit tests)



DATAPATH

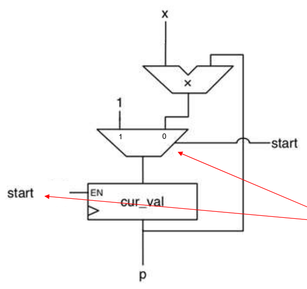
Compute
p = p * x

(repeat this **a** times)

Needs p = 1 when start=1

8

RTL Implementation (circuit + unit tests)



DATAPATH

Compute
p = p * x

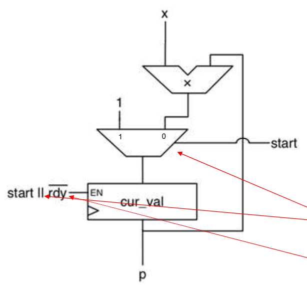
(repeat this **a** times)

Needs p = 1 when start=1

10

RTL Implementation (circuit + unit tests)

Problem: what if x changes during execution (after start=1)?



DATAPATH

Compute
p = p * x

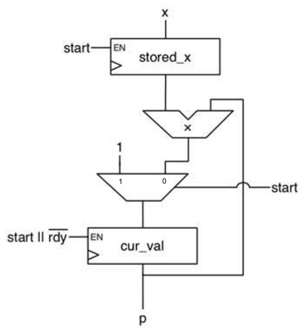
(repeat this **a** times)

Needs p = 1 when start=1

Needs p = p*x when rdy=0
(when busy computing)

11

RTL Implementation (circuit + unit tests)



Problem: what if x changes during execution (after start=1)?
Solution: add register stored_x

DATAPATH

Compute
p = p * x

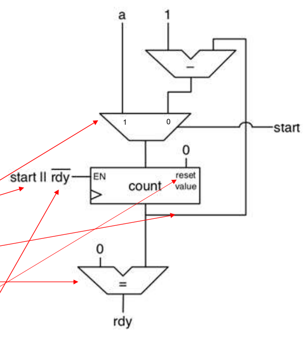
(repeat this **a** times)

Needs p = 1 when start=1

Needs p = p*x when rdy=0

12

RTL Implementation (circuit + unit tests)



CONTROL FSM

Count from **a** to 0

Need:
count=**a** when **start**=1

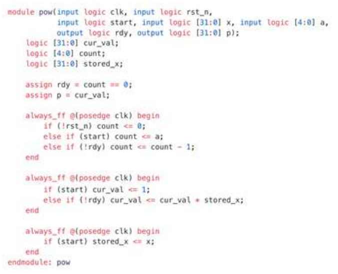
count=count-1 when **rdy**=0

rdy=1 when count=0
keeps count=0 after **rdy**=1

count=0 on reset
(ensures **rdy**=1 after reset)

13

System Verilog Implementation



```
module pow(input logic clk, input logic rst_n,
           input logic start, input logic [31:0] x, input logic [4:0] a,
           output logic rdy, output logic [31:0] p);
    logic [31:0] cur_val;
    logic [4:0] count;
    logic [31:0] stored_x;

    assign rdy = count == 0;
    assign p = cur_val;

    always_ff @(posedge clk) begin
        if (!rst_n) count <= 0;
        else if (start) count <= a;
        else if (!rdy) count <= count - 1;
    end

    always_ff @(posedge clk) begin
        if (start) cur_val <= 1;
        else if (!rdy) cur_val <= cur_val * stored_x;
    end

    always_ff @(posedge clk) begin
        if (start) stored_x <= x;
    end
endmodule: pow
```

14

Finishing Up!

- Clear-box tests for your implementation
- Verification
  - Simulate in ModelSim
- Fabrication
  - Synthesize in Quartus ➔ bitstream, post-synthesis netlist
  - Validation: test bitstream using DE1-SoC
  - Validation: simulate post-synthesis netlist in ModelSim

15

More Verilog Guidelines

1. sequential, use <= (non-blocking assignment)
2. latches, use <= (non-blocking)
3. combinational logic in always block, use = (blocking)
4. seq. and comb. logic in same always block, use <= (non-blocking)
5. do not mix = and <= assignments in the same always block
6. do not make assignments to the same variable from more than one always block
7. Use $strobe to display values that use <=
8. Do not make assignments using #0 delays

16